COMPUTER SCIENCE 312 (Winter Term 2021)
Programming Language Design
Prof. Levy

Problem Set 2

Due on github 11:59pm Friday 19 February

1 Reading Assignment: *Essentials of Programming Languages, 2nd Edition*, Chapter 1

2 Programming Assignment

Exercise 1.19, page 31. *Hint*: Use (1) the `occurs-free?` and `occurs-bound?` predicates in Figure 1 on the next page; (2) the `makeset` function on page 112 of *The Little Schemer*, and (3) a *filter* function that takes a predicate and a list and returns the list elements that satisfy the predicate: e.g.

```
> (filter odd? '(1 2 3 4 5 6 7 8 9))
(1 3 5 7 9)
```

Exercise 1.23, page 32.

Exercise 1.31, page 37. *Hint*: Represent bound variables as a list of lists, initially empty, of variables that get declared by `lambda`. As you recur, add another list to the big list. For example,

```
(lambda (a b c)
  ...
  (lambda (d e)
```

can be represented by the list `((d e) (a b c))`.

Extra credit. For extra credit, write a function `curry` that takes a lambda ression and returns its curried form. For example:

```
 > (curry '(lambda (a) (lambda (b c) (a b c))))
(lambda (a) (lambda (b) (lambda (c) ((a b) c))))
```

Hint: as with the other problems, you should use a three-way conditional:

1. `exp` is a symbol: return unmodified

2. `exp` is a lambda abstraction: return a new lambda expression with the first param of the original lambda expression as its only param. If there is only one original param, the body of the new lambda expression is the same as the original body; for more than one param, the body is created by recurring on a new lambda expression whose params are the cdr of the original params. What if the original lambda has NO params? I'll leave this one up to you!

3. `exp` is a function application: first map the curry function to all sub-expressions in the . Then build a left-associative version of the sub-expressions by (i) doing a "deep reversal" of the sub-expression list; (ii) building a right-associative version of this deep reversal; (iii) deep reversing the resulting list. A deep reversal is one where all the sub-lists of a list are reversed; e.g.,

```
 > (deep-reverse '((a b (c d e))))
(((e d c) b a))
```

It makes more sense to interpret e.g. `(f x y)` left-associatively as `((f x) y)` rather than right-associatively as `(f (x y))`, because we know that f is a function, but we don't know anything about `x` or `y`. Hence we need to implement left association rather than right association, which is harder, because of the asymmetry of car and cdr.